LEVEL II

1

FINAL REPORT VM/370 SECURITY RETROFIT PROGRAM-DETAILED DESIGN AND IMPLEMENTATION PHASE

B.D. GOLD
R.R. LINDE
M. SCHAEFER
W.M. SHASBERGER
D.H. THOMPSON
P.C. WARD

21 MAY 1978

3418

SDC-TM-6062/001/00

382100

81 12 22 105

21 May 1978

System Development Corporation
SDC- TM-6062/001/00

FINAL REPORT VM/370 SECURITY RETROFIT PROGRAM DETAILED DESIGN AND IMPLEMENTATION PHASE

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A	

B. D. Gold, R. R. Linde, M. Schaefer,
W.M. Shasberger, D.H. Thompson, P.C. Ward
System Development Corporation
Santa Monica, California 90406

Contract MDA903-76-C-0260

ABSTRACT

This report describes a design strategy for performing a retrofit to IBM Corporation's Virtual Machine 370 (VM/370) system that will provide a time-shared environment in which user processes bearing differing military classification levels may be operated simultaneously without compromise to military security. The strategy entails drawing together into a secure kernel those system functions that may be exploited to violate security. This report finalizes the results of the first two years of an ongoing research and development program.

INTRODUCTION

The VM/370 Security Retrofit Program is a research and development initiative, funded by the Defense Advanced Research Projects Agency (ARPA), with additional funding provided by the Canadian Department of National Defence and the United States Central Intelligence Agency. The program's primary goal is the security retrofit of a popular commercial operating system (VM/370). Two approaches were originally planned: (1) the design of a feasible, formally verified security kernel to VM/370 and (2) a "hardening" effort to repair known VM/370 penetration weaknesses. It was subsequently decided not to proceed with the VM/370 hardening task because of the uncertainty of the end result: correction of known security flaws does not guarantee the absence of exploitable, but not yet detected, security flaws in the hardened system.

In the first year of the research program, the feasibility of adding a security kernel to VM/370 was studied and a kernel design for the system was produced. The retrofitted system is called KVM/370 (for kernelized VM/370). The security enforcement mechanism, the kernel, must implement a reference monitor that enforces a security policy. A security kernel is a reference monitor that:

a. mediates all attempts to access security objects;

b. is protected from the tampering attempts of either the control software or the users;

c. is verifiably correct.

A security policy has been evolved that will permit as general a form of controlled sharing of machine resources and classified data as possible within the constraints of defining a kernel that will:

a. be verifiable with respect to enforcing that policy;

b. have an acceptable effect on overall VM/370 performance;

c. require minimal rewriting or replacement of existing code in the VM/370 Control Program (a retrofit);

d. preserve a maximum compatibility with VM/370 applications.

This effort has been inspired by the belief that encapsulation of multiple, individual copies of an operating system under a virtual machine monitor system can provide a practical, secure operating system. SDC's experience with IBM's VM/370 supports this belief. Even though the present implementations of VM/370 are totally insecure against planned intrusion[1], this system appears to have sufficient potential to warrant the present retrofit effort.

RETROFIT STRATEGY

A methodology was developed for partitioning the VM/370 control program (VM/370-CP) into security-relevant and non-security-relevant modules. The decision process is based on the principles of least privilege and least common mechanism, defining security-relevant code in CP as that code which executes privileged instructions or the code which accesses global system data (i.e., control blocks traversing security levels). In this way, security- relevant CP modules are directly identifiable.

It was found in the first year of the project that most system data need not be truly global, but global only over the VMs operating at a given security level. The VMs at a given security level[2] could be supported by a combination of a formally verified kernel operating in real supervisor state and a Non-Kernel Control Program (NKCP) executing in real problem state and consisting of all non-security-relevant VM/370-CP code. The NKCP would execute as a virtual machine, having access only to global system data for the virtual machines it is supporting at the given security level.

In principle, there are significant differences between a security retrofit to VM/370 and a new design of a secure VM/370. In both cases it is necessary to design and specify the security enforcement mechanisms for the security kernel, as well as to derive the set of formal security invariants the kernel must preserve over the system. It is found, however, that much of the code in an operating system that virtualizes a computer, such as that found in VM/370-CP, either has no security relevance or can be trivially modified so that it no longer has any security relevance. Hence, much of the existing code which provides functional capabilities to the virtual machines is essentially usable as it stands.

Secondly, it is observed that such code, in fact all of VM/370, can be virtualized. The strategy suggested by these observations involves designing and verifying a relatively small body of code which is just powerful enough to provide primitive virtualization and which controls all forms of I/O access with respect to the security policy. It is thus conceptually possible to run numerous copies of VM/370 atop this simple kernel, each running in virtual supervisor state. Since the kernel is the final arbiter and all access to real devices must eventually pass through it (these accesses all require invocation of privileged instructions, i.e., real supervisor state), no action of the virtualized VM/370-CP can compromise security. The users would run their programs atop the untrusted copies of virtualized VM/370. If a virtualized VM/370-CP were to attempt to perform actions contrary to the security policy, the kernel would prohibit such actions from taking place. These potential denials of service could be avoided by deleting the related code from the virtualized VM/370-CPs, but it is important to observe that these matters have no effect upon the enforcement of security since (1) the kernel was designed to enforce a specific security policy, (2) the kernel was formally verified to support the enforcement of that policy, and (3) the correctness proof of the kernel made no assumptions about any of the virtual machines running atop the kernel, particularly none with respect to an NKCP itself.

In the interest of enhancing the performance of such a kernelized system, it might be necessary to give certain system modules access to multilevel system data. These are the modules which control the sharing of real system resources among virtual machines at different security levels. In order to maintain system security, it is necessary to ascertain that such resource management modules properly utilize the privileges granted them by the added common mechanism.

Such modules become "trusted processes." Where possible and practical, the trusted processes are to be given the same formal verification the kernel processes receive.

Where this is not practical or possible[3], the trusted processes are subject to a thorough audit for the presence of errors or Trojan Horses, encapsulated into a limited address space with restricted reading and writing privileges, and restricted so that they operate in real problem state with virtual addresses. These latter processes are known as semi-trusted processes.

SECURITY POLICY

The KVM/370 kernel is designed to enforce a military security policy. This requires the preservation of two security properties, the "security condition," and the "*-property."[4] These properties are described in terms of three types of entities: subjects, objects and security levels. Subjects are the active elements of the system for which data access must be controlled (e.g., users, processes). Objects are the data or data containers, access to which must be controlled by the kernel. There is a security level associated with each subject or object which describes the degree of clearance of the subject or sensitivity of the object. A partial order, called "dominates," is defined on the security levels. Specific interpretations of these elements will be given below.

The security condition requires that no subject may access an object for the purpose of reading or updating unless the level of the subject dominates that of the object. The *-property demands that a subject may have write access to an object (permission to both read and write) only if the subject and object are associated with precisely the same security level.

In the main, subjects in KVM/370 are interpreted as the individual NKCPs. The kernel provides isolation among the NKCPs, but provides little or no additional isolation between VMs under the same NKCP beyond that already provided in VM/370. Since all VMs operating under the same NKCP act at the same security level, the kernel protects each VM from other VMs at different security levels, but not necessarily from VMs at the same level. Global processes, which must interface with several NKCPs at different levels, are also subjects.

The objects in KVM/370 are collections of data areas on DASD devices (or the entire DASD volume), tape volumes, unit record devices, real core pages and processes, and VM working environments (control blocks, scratch storage registers etc.).

During the current year, the evolution of United States National Security Policy was studied in an effort to make KVM/370 more responsive to the modifications that are being made to Executive Order 11652 and 11905. These modifications make it clear that it is essential that computer systems not divulge information to unauthorized individuals on the one hand, while prohibiting the overclassification of data on the other hand. KVM/370 enforces the *-property to prevent unauthorized declassification of data, and produces detailed historical collateral classification information for every new volume created by the system as a means of justifying its classfication as a function of the classifications of all data to which the virtual machine that created it had access. This historical classification information may then be reviewed by a security officer possessing original classification authority in order to determine the appropriate classification for the data. Details on the KVM/370 Security Policy may be found in TM-6062/230/00, 11 May 1978.

OVERALL SYSTEM ARCHITECTURE

Figure 1 represents the architecture of kernelized VM/370 (KVM/370), consisting of the following domains:

1. The kernel and verified trusted processes, executing in real supervisor state;

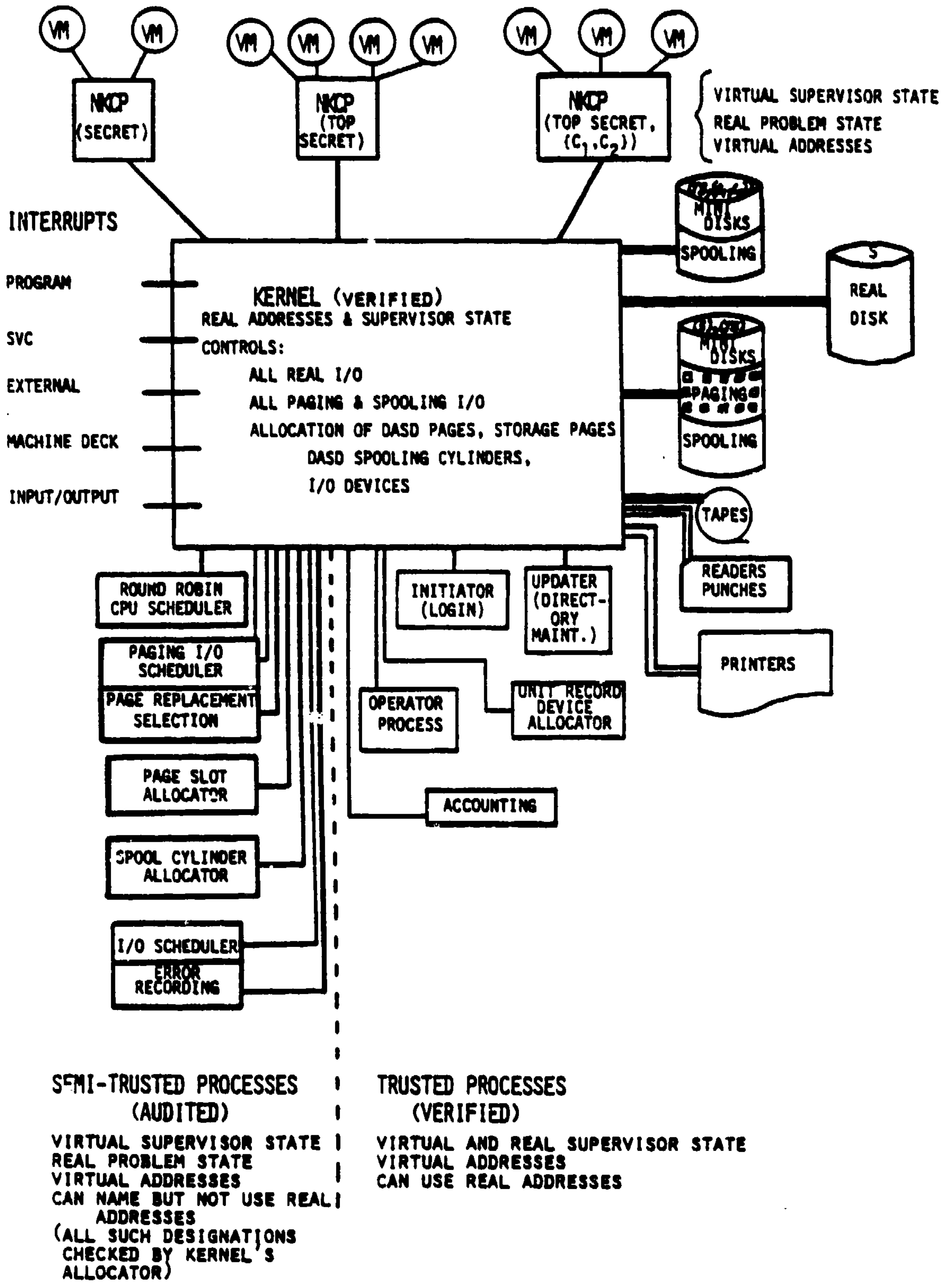

Figure 1. KVM/370 System Architecture

2. The audited semi-trusted processes, having access to some global system data, executing in real problem state, but having access only to virtual addresses;

3. The NKCPs, one per security level, having access to system data for the supported security level only, executing in real problem state, having access only to virtual addresses;

4. The user VMs, each controlled by the appropriate NKCP for its security level, executing in real problem state.

It was intended that all kernel code and trusted process code would be written in a strongly typed Pascal-based programming language such as the EUCLID[5] language in order to facilitate formal verification.

However, as the time for system implementation drew near, it was found that there were no available production quality compilers for EUCLID or any other thoroughly typed Pascal-based programming language that would permit efficient system programming on an IBM/370 base machine. The requirement that the system programming language possess the capability of addressing and manipulating IBM/370 dependent data structures is essential since the kernel must analyze, prepare, and maintain numerous tables and control blocks whose structure is dictated by the hardware. In order to provide for the future verification and certification of KVM/370, it is desirable that a maximum of detail on the manipulation of these data structures be expressed in the higher-order language rather than in assembly language. Lastly, it was essential that the compiler reliably produce highly efficient executable code, lest the performance costs of the system be impractically high. It was also necessary that the compiled code not require a run-time support package for its execution, since the run-time package could be a possible source of security compromise.

After serious consideration of numerous languages for which compilers either existed or were proposed, it was decided by ARPA that system efficiency was of sufficient importance to permit the use of a programming language that was not Pascal-based, providing it satisfied the other exigencies for the project.

The language selected for the implementation of KVM/370 was the J3 dialect of the JOVIAL Programming Language, whose optimizing compiler is in use on the AWACS project[11]. JOVIAL is not a verification-oriented programming language. Consequently, while the formal specifications of KVM/370 will be formally verified against the security policy enforcement criteria, the first implementation will not be formally verified. Subsequently, when a production quality compiler exists for a verification-oriented systems programming language KVM/370 can be recoded in that language and then verified.

DESIGN TRADEOFFS

The user's system-use expectations will have an impact on the system architecture size of the kernel and trusted processes, overall system performance, and level of effort required for implementation and formal verification. Resource scheduling and management can be performed on either a system-global or an NKCP-local basis. If done on a system-global basis, the size of the kernel and trusted processes is increased, the interfaces between the NKCPs and the global processes becomes more intricate, verification becomes more difficult and costly, system modification becomes less facile, but system performance improves. If done on a local basis with most resource management decisions performed by the NKCPs and perfunctory reconciliations performed by the kernel, the opposite results hold; system design, implementation, verification and interfaces are simplified, while system performance may be adversely affected.

In terms of greatest all-around adaptability to applications, ease of implementation and verification, and best multilevel security, we concluded that:

- DASD page areas will be global;
- Main page frame management will be based on global allocation with global page replacement;
- Multilevel shared reentrant systems will be provided with all shared pages locked into core;
- The CPU will be scheduled by the NKCPs.

KERNEL DESIGN

The kernel and trusted processes are the only portions of KVM/370 whose formal specifications will be formally verified. The system is being designed such that there are no "upward" functional dependencies (i.e., no function at level of abstraction i depends for its correct operation on any function from level of abstraction j, if $i < j$)[6] and [7]. In this way, it can be demonstrated that no trusted code depends on the correctness and non-maliciousness of any untrusted, unverified code. Further, the formal proof of correctness of KVM/370 will require only (1) the kernel and trusted processes be shown to satisfy the requirements of enforcing the security policy, and (2) a demonstration of the absence of unauthorized signalling capabilities within the semi-trusted processes.

In the case of the Start I/O request to the kernel, the entire channel control program is copied into a portion of the kernel's domain where it is protected from modification by asynchronous attack. Address translation is performed on the channel program. Then it is legality checked by the kernel to guarantee that the program performs only valid accesses, that all referenced pages are locked into core, and that the channel program is not self-modifying and contains no puns or other security threats[6]. The I/O scheduler is then invoked and control is eventually passed to the dispatcher module, simulating an I/O Fast Release. When the I/O interrupt finally takes place, the relevant pages are unlocked, and the condition code is passed as an interrupt to the appropriate NKCP.

Each security level has a unique address space for use by its NKCP. With the exception of the functions enumerated below, no NKCP can communicate outside its address space or its VMs' address spaces. Spool files, virtual channel-to-channel adapters (CTCA), and inter-VM messages are handled by the NKCP and consequently cannot violate the kernel's enforcement of the security policy.

The notable exceptions are:

- Append-up for writing machine error records and accounting data which are processed at the highest security level in the system;

- Read-down to obtain access to a DASD whose classification is dominated by the clearance of the user's VM.

For purposes of design simplicity, each NKCP will appear to be uninterruptible just as VM/370-CP currently is, i.e., the NKCP's critical regions will be preserved. The NKCP may, in practice, be interrupted by the kernel, but only if no NKCP shared variable (e.g. its set of active page frames or real addresses) is modified while it is servicing a user request. This constraint on NKCP shared variables is the result of considerable effort in the area of kernel-NKCP interaction. The consequences of this design decision, as well as the considerations that led to it, are detailed in the Appendix. An NKCP terminates a critical region (a locally uninterruptable code segment) when it either schedules a VM (issues an LPSW) or when it issues an I/O request.

NKCP DESIGN

Code is security-relevant if it can influence unmediated I/O directly through the use of privileged instructions that manipulate devices or user domains, or indirectly through the use of data structures that either contain security enforcement data or can be viewed and modified by processes operating at different security levels. Data is security-relevant if it contains global information which traverses several security levels.

It appears that a considerable amount of CP code is security-relevant because it makes wide use of global system tables. Many of these tables could be distributed so that each copy contains only data relevant to a unique security level. The code manipulating these distributed tables could easily be made reentrant so that most of the code could lose its security relevance (privileged instructions would still be security-relevant, however.)

An alternate approach is driven by identifying non-security-relevant code in CP and virtualizing it out of the privileged execution domain. The remainder, plus additional security enforcement code, becomes the kernel. The more code that is virtualized, the less there is to verify. Apparently, the efficiency of this system degrades as code is virtualized out of the kernel.

SYSTEM VERIFICATION

Formal verification of kernel and trusted processes at the specification level will be of two kinds. These functions will be shown to correctly implement the sharing policy between subjects and objects in terms of the basic security principle and the *-property. This form of proof involves demonstrating that all system state transitions preserve a set of security invariants. The proof of correctness will be achieved with the assistance of an automated verification tool which enhances the credibility of the formal logical demonstration. The second phase of the formal verification process is formal proof that the trusted state transition functions themselves obey the *-property with respect to the system objects they read and write. The analysis techniques required for this verification involve simulated symbolic execution of source code. Research into methods of flow analysis is being conducted by Millen[12], Denning[8] and others. In the latter work, consideration is being given to the use of a compiler to verify compliance with the security policy by the source program. In Millen's work, the flow analysis is applied directly to the top-level specification. Both methods involve the use of automated tools with which the flow analysis is performed, followed by the manual imposition of the security policy to the data flow paths in order to establish a set of relations which must be shown to be preserved by the system. The flow analysis phase of KVM/370 verification has been postponed until such time as appropriate tools are available to the project.

POSSIBILITY OF HARDWARE ERRORS

One of the problems considered by the project was the possibility of violations of the security policy occurring because of failure of the hardware security controls. Some possibilities considered were:

- failure of the privileged operation protection mechanism;
- an error in address translation or the Translation Lookaside Buffer (TLB);
- failure of the storage protection mechanism;
- mis-interpretation of a CCW by a channel;

. an I/O device responding to the wrong device address;

. mishandling of a command by an I/O device.

Errors in the operation of the CPU and inboard channels were considered unlikely if the system receives proper maintenance. In addition, it appears to be difficult to guard against this type of failure. For similar reasons, we decided to ignore the possibility of an I/O device responding to the wrong address because address recognition logic on the S/370 channel is fairly simple. This left us with the possibility of an I/O device mishandling a channel command. The most probable case of this type seemed to involve seeks on moving head devices: the possibility of a mechanical error moving the access arm to the wrong cylinder.

Questions had arisen as to the possibility that certain Direct Access Storage Devices were liable to incorrect seeks with a frequency that increased with their age. SDC conducted an investigation to establish hard data on the reality of these reported threats. Investigation of the logic of 3330 type devices showed a vanishingly small probability of a missed seek, inasmuch as the device controller counts the tracks electronically as they pass under the head. We also forced over 20,000 seeks of 350 cylinders and tested for an incorrect home address after each one. We found no missed seeks, nor were any seek retries reported in the error log for that day. We now believe that the redundancy checking built into the 3330 provides sufficient reliability for multilevel security applications.

Obviously, if a DASD accesses the wrong cylinder on a seek command, it is possible for a malicious user to read whatever is on the cylinder accessed, or to write incorrect information into those records. If the probability of a missed seek going undetected by the controller is as high as 0.1%, a user who causes a large number of seeks can reasonably expect to gain unauthorized access to data belonging to other users several times a week.

The results of the study of the 3330 have obviated the necessity of having to limit each DASD volume to a single level of security, as had been contemplated prior to the experiment. However, some installations may use older or other direct access storage devices which may not be as reliable as the IBM drives that we tested. As a result we decided to provide a mechanism for protecting against mis-seeks, albeit at some cost. I/O requests are separated into paging/spooling requests and all others.

We note that all modern DASD devices can have an 8-byte key added to each paging block without affecting the number of pages which will fit on a cylinder. It was decided to write in each key the real cylinder number, track and record number, VMid and virtual page number that it contains, encrypted by a key that is determined at system startup and unique for each security level. This should make the task of the would-be penetrator hard enough to discourage any attempt to use this mechanism.

For general I/O, it was decided to include a read-home-address CCW after each seek, and to have the kernel validate the home address on completion of the channel program. This would increase the average time required to execute a channel program by 1/2 revolution (about 8 ms.). Since this represents nearly a 100% overhead in I/O operations, it was decided to partition devices into two classes: trusted devices and untrusted ones. A device is considered trusted if (1) it has been designated by the installation's security officer as trusted, (2) no mis-seeks on that device have gone undetected by the hardware (these usually result in unit-check with no-record-found), and (3) less than some threshold number of hardware-detected mis-seeks have occurred. If any of these conditions is not satisfied, the device is regarded as untrusted. Home address verification is applied only when (1) the device is regarded as untrusted, (2) the I/O is not for paging or spooling, and (3) the I/O has been requested by an untrusted process (NKCP or scheduler/allocator).

ELIMINATION OF KNOWN SECURITY FLAWS

Almost every security flaw in the VM/370 system involves the input/output functions[1]. Since there is no address space validation of input/output by the hardware, other than that performed by the storage protection keys, VM/370 must validity check all channel programs and relocate all virtual addresses. This includes both main storage addresses, and DASD cylinder addresses in seek arguments and home addresses. The same I/O logic is repeated for several different requirements: virtual spooling support, virtual console support, virtual channel-to-channel adapter support, and a special VM/370 I/O interface. Each variation of this support means that errors may be present.

These errors occur in the translation of channel programs as a result of the complexity of the channel command language. For example, the same word in a channel program might be used as a command or as an operand address depending upon the execution sequence of the program[1]. Since the System/370 architecture allows puns in the channel, in that a word's interpretation depends on whether it is received as the leading or trailing portion of a long command, it is possible to bypass checking in these modules and access DASD records [1].

Under KVM/370, we will not allow channel command words to take on different meanings depending on the sequence of execution. Primarily, this means that an NKCP cannot submit certain channel commands with transfers or with certain modifier bits set. This does not preclude users (VMs) from constructing such channel programs, it merely requires the NKCP to put them into a standard form before submitting them to the kernel. Further, these commands will be copied into the kernel's data space and translated and modified there, preventing their modification by an NKCP between the time of translation and time of execution. Also, self-modifying channel programs will not be permitted, such as those used by OS/360 ISAM.

Certain VM/370 penetrations[1] dependent upon simultaneous input/output and CPU execution are being countered by removing from the address space of the requestor all pages which are buffering input. This applies to both NKCP and user VMs. In the event either NKCP or a VM needs access to such a page, its execution will be delayed until the I/O completes and the page is made available. For example, under VM/370, careful timing of asynchronous execution could be used to exploit a bizarre oversight in condition-code checking to gain a total system penetration (real supervisor state)[1].

Although the treatment of storage and timing channels[9] is beyond the scope of a system such as VM/370, they must be controlled in a military system such as KVM/370. In this respect, we will thoroughly audit all semi-trusted processes that allocate and schedule resources among VMs at different levels for Trojan Horses. Hence, it will not be possible to transmit information over a covert communication channel at a high enough bandwidth to make such attempts worthwhile.

Lastly, the initiator (Logon) will not allow a user process to assume its identity (masquerade) at an unattended terminal. Under VM/370, it is possible to reveal one's password to a virtual machine masquerading as CP[1]. This can happen when users must share terminals, a frequent occurrence at most installations. The initiator will require that a unique character string be submitted at LOGON and will monitor all input lines for this string. Strings containing passwords will thus be processed by the initiator rather than by a virtual machine.

CURRENT STATUS

The feasibility of performing a VM/370 security retrofit was demonstrated during the first year of project activity. The results of that work were reported in the TM-5855 series, and included an informal system design identifying the major security kernel functions. The input and output parameters were defined and their effects on KVM state variables were described.

During the year just ended, the security kernel and trusted processes were formally specified in the SDC specification language, INA JO, a strongly-typed dialect of the first order predicate calculus. After the system data bases were defined, the coding of the NKCP and semi-trusted processes was begun. In the last quarter of the project, the implementation of the kernel and trusted processes was begun in the J3 dialect of JOVIAL.

KVM/370 security policy was re-examined in light of proposed modifications to the National Security Policy as defined in Executive Orders 11652 and 11905. KVM/370 security policy now takes account of both discretionary and non-discretionary aspects of security.

The remaining documents in this TM-6062 series describe in greater detail the results of these activities. The interested reader is directed to the series table of contents, found in TM-6062/000/00, 21 May 1978.

PLANS

It is expected that system testing and integration will conclude by the fall of 1978. At that time, KVM/370 will be installed in a testing environment within the Defense Communications Agency Engineering Center. Here, the prototype system will be evaluated on a set of selected benchmark workloads and its perfomance will be tuned to the extent possible within the constraints of the security policy. In this way, the first steps can be undertaken toward determing the costs of multilevel security on an IBM/370 mainframe. Initially, test cases will be run under varying conditions in a periods processing environment as expressed in TM-5855/003/00, 21 May 1977. This will establish a basic scale against which the operation of KVM/370 can be judged. These will be followed by selected KVM/370 runs that approximate the periods processing approach: one virtual machine (one color); two virtual machines (two colors), etc. Various test case workloads will be defined and specific measurements will be performed. Acceptance of KVM/370 will be made by relating dollar costs to run time, and by an evaluation of the periods processing approach as it impacts user requirements (see TM-5855/003/00).

CONCLUSION

In this paper, we have presented a design strategy for performing a retrofit to VM/370 which will provide a multilevel secure operating environment. The strategy is heavily based on the principles of least privilege and least common mechanism. The research and development activities described in this paper transpired in the period March 1977 through May 1978. The implementation of KVM/370 is currently in progress and it is anticipated that a prototype version of the system will be installed within the Defense Communications Agency in the fall of 1978. Further results will be reported in a future paper.

ACKNOWLEDGEMENTS

The authors wish to acknowledge the contributions and suggestions made by the following individuals: E. Book, M. Branstadt, E. Burke, J. Fraley, T.H. Hinke, D. Hollingworth, A.K. Jones, H. C. Lauer, R. Lyons, C.A. Melkerson, M. Moriconi, M. Orceyre, G. Popek, G. Schroeder, W.M. Shasberger, S. Walker, C. Weissman, J.H. Yott, and D. Zucker. We would also like to extend our sincere appreciation to our technical editor, B.L. Sargeant, and N. Spies, who provided typing support.

----

[1] C.R. Attanasion, P.W. Markstein and R.J. Phillips, Penetrating an Operating System: a Study of VM/370 Integrity," pp. 102-116, IBM SYSTEMS JOURNAL, vol. 15, no.1, International Business Machines Corp., 1976.

[2] A security level (C, K) consists of a hierarchical classification C from the ordered set {unclassified, confidential, secret, top secret}, and a category K consisting of a subset (possibly empty) of the set of special access compartments.

[3] The semi-trusted processes serve as schedulers and allocators of global resources and have the potential to be used as an illicit signalling path in violation of the Mitre *-property by modulating the global state variable, TIME. There is no known method for formally demonstrating that an algorithmically correct, Trojan Horse free, scheduler cannot be manipulated by users in such a way that the users can cause clock time to become a signal to other users.

[4] D.E. Bell and L.J. LaPadula, "Secure Computer Systems: A Refinement of the Mathematical Model," MTR-2547 vol III, the MITRE Corporation, Bedford, Massachusetts, 28 December 1973.

[5] B.W. Lampson, J.J. Horning, R.L. London, J.G. Mitchell, and G.J. Popek, "Report On The Programming Language Euclid," Xerox Research Center, University of Toronto, USC-ISI and UCLA respectively, December 1976.

[6] P. Janson, "Using Type Extension to Organize Virtual Memory Mechanisms," MIT/LCS/TR-167, Massachusetts Institute of Technology, September, 1976.

[7] D.P. Reed, "Processor Multiplexing in a Layered Operating System," MIT/LCS/TR-164, Massachusetts Institute of Technology, June, 1976.

[8] D.E. Denning, "Secure Information Flow in Computer Systems," PhD. Thesis, Purdue Univ., Computer Sci. Dept., West Lafayette, Indiana, May 1975.

[9] B.W. Lampson, "A Note on the Confinement Problem," COMMUNICATIONS OF The ACM, October, 1973, pp. 613-615.

[10] D.E. Bell and L.J. LaPadula, "Computer Security Model: Unified Exposition and Multics Interpretation," ESD-TR-75-306, The MITRE Corporation, Bedford, Massachusetts, June 1975.

[11] AWACS Jovial Staff, "Jovial User Manual for AWACS J3," TM-WD-751/250/00J, July 12,1976.

[12] J. Millen, Security Verification in Practice, COMMUNICATIONS OF THE ACM, May 1976, Vol. 19, Number 5, pp. 243-250.

APPENDIX A

A SOLUTION TO THE "LOAD REAL ADDRESS" PROBLEM

BACKGROUND: During the first year of the KVM/370 project, attention was paid to what is known as the "Load Real Address" problem. This problem is concerned with the fact that an NKCP needs to be able to "locate" certain pages of the VMs under its control. This is handled in VM/370-CP by the Load Real Address (LRA) instruction. The LRA may be used for channel program translation, or to locate the operand(s) of a privileged operation that CP is simulating.

The problem which arises in KVM/370 is that the decision to "steal" a page is made by a global process, not under control of the NKCP. Consequently, there is no guarantee that the page, once "located" by the NKCP, will stay at the same real address, or even remain in main storage, long enough to be used. In order to avoid frequent and embarrassing denials of service, it is necessary to guarantee that a virtual page stays in the same place from the time the NKCP has been given its real (or "real") address, until the NKCP is no longer relying on the address given for that page. The following discussion expands on the complexities of the problem and presents what is believed to be a solution to it.

RELATED CONSIDERATIONS: During the year several points of view have been held concerning real addresses. It would probably simplify the kernel-NKCP interface if the NKCP were allowed to access the real addresses of the pages under its control. On the other hand, there are several high bandwidth data channels involving real page addresses. The current design calls for the NKCP to gain access to pages containing operands by having them placed in its own address space (The kernel inserts their real addresses in the NKCP's page table). This allows the NKCP to read and/or modify data for instructions that it simulates for its VMs, without knowing the real addresses of the pages. For channel program translation, the NKCP will leave virtual addresses in the IDAW lists, which the Request-I/O handler will translate to real addresses.

In order to simplify the kernel's handling of process scheduling, it was decided to treat NKCPs as logically non-interruptable. The kernel will refrain from presenting the NKCP with interrupts during its operation (just as VM/370-CP is designed to run with interrupts disabled). The kernel will also refrain from running any other NKCPs until the current one signals the end of its critical region.

THE SOLUTION: The operation of an NKCP is considered as a critical region from the time it is entered until it dispatches a VM or relinquishes the CPU. Any interrupts taken by the kernel during this period will be handled to the extent possible by the kernel and schedulers, but no other NKCPs will be dispatched, even if the current NKCP's time-slice ends. Interrupts requiring action by any NKCP, including the current one, will be stacked until the end of the critical region. Any pages swapped in at the NKCP's request or to which the NKCP gains access (by "attach page") will be placed under a temporary lock which prevents their page frames from being stolen during the critical region. The critical region (and temporary lock) will end when the NKCP makes either a Dispatch-VM or a Release-CPU kernel call. If an NKCP requires that a page be at a fixed address for a longer period of time, it must make an explicit Lock-Page call.

Note that such locks (either temporary or long-term) cannot be attached to a page which is not present or which is being used for a conflicting purpose. For example, a page that has been stolen and is being swapped out cannot be locked unless the requester can reclaim the page (this ability is not supported by the current KVM/370 design). A page which is being used for the buffering of input cannot be locked for CPU usage or output, nor can a page being used for the buffering of output be locked as an input buffer.

This approach has the following consequences:

(1) No page will be stolen from an NKCP or its VMs except when the NKCP is running a VM or waiting for an interrupt. (The latter case is equivalent to DMKDSP loading a wait-state PSW because there is no work to do). VM/370-CP always checks page status when a new request arrives from a VM by doing a LRA and/or calling DMKPTRAN. Similarly, NKCP does not rely on page locations remaining constant across such operations, so stealing a page cannot cause the NKCP to make an erroneous assumption.

(2) If VM/370-CP requires that a page keep the same real address after a call to the dispatcher, it explicitly requests DMKPTR to lock the page. Thus, if the NKCP requires a page to stay in main storage across Dispatch-VM or Release-CPU calls, it must explicitly request a lock on that page. Otherwise the kernel or Select routine will be permitted to steal the page if it is the "best" page to steal.

(3) Since the kernel will call other NKCPs (that is, by invoking the CPU scheduler) only when the current process makes a Dispatch-VM or Release-CPU call, there can be no "surprise" loss of the CPU to another NKCP. This means that the kernel need not save and restore the registers in the KPROCBLOK for each process. Instead, a single level of register storage will suffice (for G-regs and timers only, since neither the kernel nor the trusted/semi-trusted process will use the F-regs). Each NKCP must be written so that it saves its own registers whenever it makes a call that invokes another process or ends its critical region. [This is not directly related

to the LRA Problem, but simplifies the design of KPROCBLOKs and the management of space for KPROCBLOKs and KVMBLOKs].

LOCKS: The kernel provides three types of locks to NKCPs.

1. A temporary lock is attached to each page which is swapped in at the request of an NKCP or to which the NKCP is given access as a result of an attach-page request. The lifetime of the temporary lock is the NKCP's critical region (i.e., until the NKCP releases the CPU or dispatches a VM). The temporary lock prevents the page from having its frame stolen; the NKCP may release the page, however, which cancels the temporary lock.

2. An I/O lock is attached to each page used in an I/O request. The page must be in main storage when the request-I/O call is made. The lifetime of the lock is concurrent with the I/O request (the lock is released when the requested I/O operation completes). The I/O lock can be cancelled only by cancelling the I/O operation.

3. A long-term lock is attached to a page at the request of any NKCP with access to that page. The long-term lock is permanent until cancelled by a specific request. Only the NKCP which requested the lock can cancel it. The kernel calls Lock-Page and Unlock-Page will be provided for this purpose. This type of lock can be used by an NKCP in response to an operator "LOCK" command or while gathering multiple pages (e.g. for an I/O operation) to insure that pages obtained earlier do not get swapped out while obtaining other pages.

All three types of locks protect the page from being stolen until the NKCP is finished with them. The second and third types of locks also prevent the NKCP from releasing the page until the lock has been cancelled.

If the SELECT routine (a semi-trusted process) attempts to select a page for which a lock exists, it will be re-entered to select another page. The kernel will refuse to steal a frame from a locked page. If the NKCP attempts to release a locked page (via release-page) or to swap out such a page, the result depends on the type of lock(s) attached to the page. If only a temporary lock is attached to the page, the temporary lock will be released and the request honored. If an I/O lock or a long-term lock is attached to the page, the request will be denied.

APPENDIX B

A GENERAL COUNTERMEASURE FOR QUOTA-TYPE LEAKAGE PATHS

The allocation of objects from a global pool of finite size allows use of that limited size as a communication path for covert transmission of data. The sending process repeatedly requests resources from the pool until a request is denied. At that point the sender knows the pool is exhausted and can release the CPU and allow other processes to run. The receiver requests a few objects from the pool, then releases them. The sender releases a large number of objects into the pool to send a one, or exhausts the pool to send a zero. The receiver receives a one or zero depending on whether its requests are satisfied. Other processes may introduce noise by exhausting the pool with legitimate requests or releasing objects they no longer need. Such noise can be filtered out via normal redundancy techniques. The state of the pool is a variable shared between sender and receiver, creating a storage channel whose bandwidth is dependent on the frequency with which such requests can be made.

Analysis of the preliminary design of KVM/370 reveals a number of such resource pools:

. Disk Pages (Page Slots)

. Main Storage Pages (Page Frames)

. Spool Cylinders

- Temporary Disk Cylinders
- Kernel Table Entries
- Kernel Storage for Dynamic Creation of Tables

A number of countermeasures have been adopted to control the use of these pools as communication channels. [1] Prediction is used on page slots and entries in the KVMTABLE and PROCESSLIST. Whenever a user attempts to Log In (a relatively infrequent event), a check is made to determine whether the necessary page slots and table entries are available. The user is denied access if they are not. [2] Temporary disk cylinders are subpooled; each security level has a private pool from which it makes allocations. No global pooling of TDisk is provided. [3] Requests for main storage pages and spool cylinders are never refused. The satisfaction of a request for a page frame or spool cylinder is reported by an interrupt which may occur immediately or after an arbitrary period of time. This converts a potential storage channel into a timing channel and lowers the bandwidth. (A process which exhausts the pool is unable to free the entries it has requested until the necessary I/O has been performed).

However, some types of requests for kernel tables cannot be predicted, and their satisfaction is not dependent on I/O.

Further, subpooling kernel storage by security level would be extremely wasteful of main storage which is a precious resource. The communication channels involving these quotas are being tolerated but restricted in bandwidth. Whenever a process request is refused because of exhaustion of a kernel table or storage pool, a return code is provided indicating to the process that some quota has been exhausted (without specifying which one). After that the process will not be permitted to make another such request for a period of time. Until that time period is over, any request by that process depending on such a quota will be denied without checking the resource pool and the return code will indicate "too soon." In this way, the communication channel is limited to one bit per time period. By setting the time period to .1 second, these communication channels are restricted to 10 bits per second.

This technique can be used on any system that has a real-time clock and can be used on any resource pool. It can be applied instead of or in addition to other countermeasures for control of quota-type data channels.